(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 May 2001 (25.05.2001)

PCT

(10) International Publication Number
WO 01/37095 A1

(51) International Patent Classification[7]: G06F 11/30, 12/14, 15/173, 15/177, H04L 9/00, 9/32

(21) International Application Number: PCT/US00/31032

(22) International Filing Date:
10 November 2000 (10.11.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
132916 14 November 1999 (14.11.1999) IL
09/561,395 28 April 2000 (28.04.2000) US

(71) Applicant: CLICKNET SOFTWARE, INC. [US/US]; 2460 Zanker Road, San Jose, CA 95131 (US).

(72) Inventors: HOLLANDER, Yona; 21327 Glenplace Dr., #7, Cupertino, CA 95014 (US). RAHMAN, Ophir; 1570 Ben Roe, Los Altos, CA 94040 (US). HORVITZ, Oded; Hagana St. 43, 46325 Herzelya (IL).

(74) Agents: BEATON, Glenn, K. et al.; Gibson, Dunn & Crutcher, Suite 4100, 1801 California Street, Denver, CO 80202 (US).

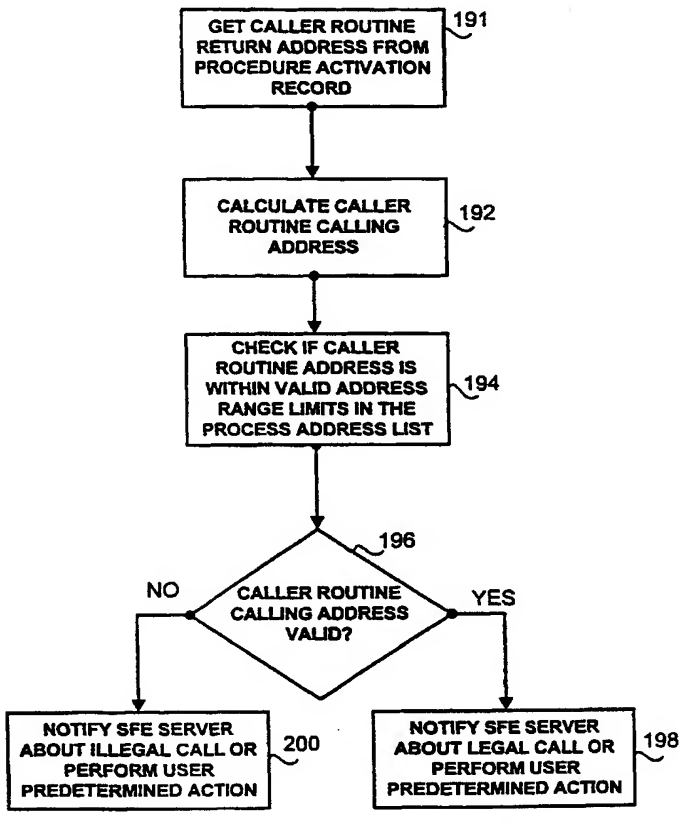
(81) Designated States *(national)*: AU, CA, JP.

(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR).

Published:
— *With international search report.*
— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: METHOD AND SYSTEM FOR INTERCEPTING AN APPLICATION PROGRAM INTERFACE



(57) Abstract: In a computer system running an operating system platform, having an operating system including a kernel space and a process space, said process space including a user application running in process space, said user application operative to intercept system calls, a method of secure function execution, said method comprising the step of examining said intercepted system call (191, 192) validity by comparing said intercepted system call originating address with range of process valid addresses (194) associated with said process from which said intercepted system call originated (191, 192) and providing notification (198, 200) as to the validity (196) of said intercepted system call (191, 192) or terminating said intercepted system call.

WO 01/37095 A1

METHOD AND SYSTEM FOR INTERCEPTING AN APPLICATION PROGRAM INTERFACE

FIELD OF THE INVENTION

The present invention relates generally to a method for detecting and preventing unauthorized or illegal access attempts within a computer system. More specifically, the present invention relates to a method for detecting and preventing attempts to exploit the buffer overflow-related weakness within a computer system.

BACKGROUND OF THE INVENTION

This application is related to Israel Patent Application Number _____ "METHOD AND SYSTEM FOR INTERCEPTING A APPLICATION PROGRAM INTERFACE" filed 14 November 1999.

Modern computers are designed with the requirements of high-level languages in mind. The most essential technique for structuring computer programs introduced by high-level languages, is the procedure or the function.

Procedures or functions are computer programs. A procedure call or a function call is a high-level abstraction that alters the flow of the calling program execution. In contrast with the more traditional "jump" or "goto" instructions, which also alter the flow of execution, a procedure or a function, after the execution of its own code, returns control to the instruction immediately following the call. To implement

procedure or function calls in the manner described, a memory device called a stack is utilized.

A stack is a contiguous block of memory containing data. Its size is dynamically adjusted by the operating system routines at run time. The data is inserted to and removed from the stack by Central Processing Unit (CPU) utilizing Assembler language instructions such as "push" or "pop".

The stack consists of logical stack frames or Procedure Activation Records that are inserted into the stack when a function is called and removed from the stack when the said function returns control to the calling program. The stack frame itself contains parameters to the called function, local variables, pointers to recover the previous stack frame, and the return address of the calling computer program. The return address is the instruction pointer of the calling program at the time of the function call.

Induced buffer overflow or buffer overflow attack is known in the art. Buffer overflow attacks exploit the lack of bounds checking on the size of input being stored in a buffer array. Arrays are predefined allocated memory devices within a computer system. By writing data intentionally past the end of an allocated array, an attacker can make arbitrary changes to data stored adjacent to the said array. The most

common data structure to be corrupted in this fashion is the stack. Therefore this type of attack is also known as stack smashing.

The prevalent form of buffer overflow exploitation is to attack buffers allocated on the stack. Such attacks attempt to achieve two mutually dependent objectives. One such objective is inserting an attack code in the form of an executable binary code native to the attacked machine. Another such objective is to change the return address to point to the attacker's supplied code now residing within said stack memory. Such attacker's supplied code may be utilized to gain enhanced privileges over said computer system.

The programs that are attacked using this technique are usually high privilege utilities or daemons that run under the user-id root to perform essential services. The effect of a successful buffer overflow attack is to provide the attacker non-authorized root privileges. Gaining root privileges within a computer system allows non-authorized users access privileged resources.

As the maximum length of the overflowing data string can be only the current depth of the stack, the inserted attack code should be short in terms of code length. Writing data outside the stack limit will result in an exception condition that will prevent the attack code to execute. Therefore, the buffer overflow attacker will be forced to write

short code and will have to use high-level System calls or Library calls. Such calls will later be utilized to gain non-authorized enhanced privileges to access privileged resources.

Several strategies, which attempt to resolve the buffer overflow weakness, are known in the art. One such strategy is to design a compiler designed to prohibit a computer program from writing past a stack segment array. Another strategy is to detect buffer overflow vulnerable programs off line and alert the user to the possibility that the system privileges may be compromised.

Another known strategy is using a repair program. The repair program can repair or fix those vulnerable programs that can be used to exploit the buffer overflow weakness.

None of the above provide a method and apparatus for prevention of buffer overflow through controlled execution of system or other calls within a computer system.

SUMMARY OF THE PRESENT INVENTION

Thus, there is a long felt need to provide a for detecting and preventing unauthorized or illegal access attempts within a computer system. More specifically, a method for detecting and preventing attempts to exploit the buffer overflow-related weakness within a computer system by validating system or other calls made within a computer system.

It is therefore the object of this invention to provide a method for preventing induced buffer overflow attack by preventing execution of high-level System calls, Library calls, Application Program Interface call and the like when such calls are illegally made.

It is therefore another object of the present invention to provide a method for preventing induced buffer overflow attack by preventing execution of high-level System calls, Library calls, Application Program Interface call and the like when such calls are made from unauthorized areas within a computer system.

It is yet a further object of the present invention to provide a method for preventing induced buffer overflow attack by preventing execution of high-level System calls, Library calls, Application Program Interface call when such calls are made from outside the user process associated with said called system or other call.

It is therefore provided in accordance with a preferred embodiment of the present invention a method of secure function execution within a computer system running an operating system platform, said operating system including a kernel space and a process space, said process space including a user application running in process space, said user application operative to intercept system calls, said method comprising the step of examining said intercepted system call validity by comparing said intercepted system call originating address with range of process valid addresses associated with said process from which said intercepted system call originated and providing notification as to the validity of said intercepted system call, or terminating said intercepted system call.

It is further provided in accordance with a preferred embodiment of the present invention a method of secure function execution within a computer system running an operating system platform, said operating system including a kernel space and a process space, said process space including a user application running in process space, said user application operative to intercept system calls, said method comprising the step of examining said intercepted system call validity by comparing said intercepted system call originating address with range of process valid addresses associated with said process from which said

intercepted system call originated and responsive to process creation inserting application program interface interception module into said created process and responsive to process creation updating process valid addresses table or responsive to process termination updating process valid addresses table.

In accordance with yet another preferred embodiment of the present invention there is provided a method of secure function execution within a computer system running an operating system platform, said operating system including a kernel space and a process space, said process space including a user application running in process space, said user application operative to intercept library calls, said method comprising the step of examining said intercepted library call validity by comparing said intercepted library call originating address with range of process valid addresses associated with said process from which said intercepted library call originated and providing notification as to the validity of said intercepted library call or terminating said intercepted library call.

In accordance with yet another preferred embodiment of the present invention there is provided a method of secure function

execution within a computer system running an operating system platform, said operating system including a kernel space and a process space, said process space including a user application running in process space, said user application operative to intercept library calls, said method comprising the step of examining said intercepted library call validity by comparing said intercepted library call originating address with range of process valid addresses associated with said process from which said intercepted library call originated and responsive to system call loading dynamic link library hooking and patching library routines associated with said dynamic link library and responsive to system call unloading dynamic link library updating process valid addresses table.

In accordance with another preferred embodiment of the present invention there is provided a method of secure function execution within a computer system running an operating system platform, said operating system including a kernel space and a process space, said process space including a user application running in process space, said user application operative to system and function calls, said system or function call intercepted, said method comprising the steps of receiving caller routine return address from said process

memory device, determining whether caller routine address is valid by comparing said caller address routine with process valid address table and providing notification as to the validity of said caller routine return address or performing user predetermined acts associated with said validity of caller routine address. The same method further comprising the step of determining said caller routine calling address by determining the address preceding said caller routine address.

In accordance with yet another preferred embodiment of the present invention there is provided a method of secure function execution within a computer system running an operating system platform, said operating system including a kernel space and a process space, said process space including a user application running in process space, said user application operative to system and function calls, said system or function call intercepted, said method comprising the steps of receiving caller routine return address from said process memory device, determining whether caller routine address is valid by comparing said caller address routine with associated process stack address area and providing notification as to the validity of said caller routine return address or performing user predetermined acts associated with said validity of caller routine address. The same

method wherein the step of receiving caller routine return address from said process memory device further comprises the step of determining said caller routine calling address by determining the address preceding said caller routine address.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitutes a part of the specification, illustrate preferred embodiments of the invention and, together with the description, serve to explain the principles of the invention:

Fig. 1 is a block diagram of the Secure Function Execution System environment generally referenced to as system 100;

Fig. 2 is a high-level flow diagram of the Secure Function Execution Server 116 operation;

Fig. 3 is a high-level flow diagram of the operation of the Secure Function Execution Server initialization module referred to in Fig. 2;

Fig. 4 is a high-level diagram of Secure Function Execution Server or the like response to an intercepted system call referred to in Fig. 2;

Fig. 5 is a high-level flow diagram of the operation of the

Secure Function Execution Server and the like library call response module referred to in Fig. 2;

Fig. 6 is a high-level flow diagram of the operation of the Calling Address Validation Routine module;

Fig. 7 is a high-level flow diagram of the Calling Address Validation Routine module relating to an another embodiment of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

The present invention overcomes the disadvantages of the prior art by providing a novel method, which detects if an attempt to exploit the buffer overflow weakness is occurring by validating use of system or other calls within a computerized system.

Reference is now made to Fig. 1, there is provided a schematic illustration of the system environment wherein the Secure Function Execution System is operating, generally referred to as system 100, in accordance with a preferred embodiment of the present invention.

The present invention is related to Israel Patent Application Number XXXXXXX. "METHOD AND SYSTEM FOR INTERCEPTING A APPLICATION PROGRAM INTERFACE" filed 14 November 1999.

As further described in detail in Israel Patent Application Number XXXXXXX, system 100 of Fig. 1 may comprise of four components of the Secure Function Execution System;

a) Secure Function Execution Server 116 is an active component. Secure Function Execution Server 116 is the

operational center of the Secure Function Execution System 100. Secure Function Execution Server 116 loads and controls System Call Interception Component 124, loads and controls API Interception Module 134, 140, and 146, responds to diverse system and library calls and acts as an interface towards the user. The Secure Function Execution Server 116 is loaded into the user space memory device 112 of a computer system. Secure Function Execution Server 116 incorporates the API Interception Control Server operations that were described in detail in Israel Patent Application No. XXXXXXX.

b) API Interception Module 134, 140, 146 and the like are active components. API Interception Module 134, 140, 146 and the like operations are described in detail in Israel Patent Application Number XXXXXXX. API Interception Module 134, 140, 146 and the like consist of Dispatch Routine, Depatch Routine, Hook and Patch Routine, Pre-Entry Routine, and Post-Entry Routine. The operations of the said routines are also described in detail in Israel Patent Application No. XXXXXXX.

c) System Call Interception Component 124 is an active component. System Call Interception Component 124 operation is described in detail in Israel Patent Application No. XXXXXXX.

d) API routine 132, 138, 144 and the like are passive components. API routines 132, 138, 144 and the like are potential objects upon which Secure Function Execution System 100 might operate. API routines 132, 138, 144 and the like are loaded into process address space memory device 118, 120, 122 and the like in user memory device 112 of system 100.

System 100 previously described in Israel Patent Application number XXXXXXX serve as model for a Secure Function Execution System in which the present invention is operative. It will be appreciated by those skilled in the art that the present invention may operate under various similar systems and that the system shown herein is an example to further illustrate the working of the present invention.

Referring to Fig. 2 there is provided a high-level flow diagram of the Secure Function Execution Server 116 operation.

Secure Function Execution Server 116 was previously described in detail in Israel Patent Application Number XXXXXXX. The operation of said Secure Function Execution Server 116 would now be briefly explained.

Secure Function Execution (SFE as it will be abbreviated from this point on in the text of this document) Server 116 initializes the application in step 150. Consequently SFE Server 116 commences its run-time operation in step 152 by constantly monitoring system calls made by diverse applications that run in the host operating system (step 152) and responding appropriately to the said system calls (step 154) as described in detail in Fig. 4. SFE Server is also constantly monitoring library calls made by diverse application that run in the host operating system (step 156). SFE Server responds appropriately to the said library calls (step 158) as described in detail in Fig. 5.

Referring now to Fig. 3 there is provided a high-level flow diagram of SFE Server 116 start-up operation referenced as step 150 of Fig. 2. SFE Server 116 start-up operation was previously described in detail in Israel Patent Application Number XXXXXXX.

First SFE Server 116 loads System Call Interception Component 124 into kernel space memory device 114 (step 184). After

establishing communication with System Call Interception Component 124 SFE Server 116 queries System Call Interception Component 124 for the list of active processes 118, 120, 122 and the like (step 186). Using the said list of active processes 118, 120, 122 and the like SFE Server 116 creates a list of valid address ranges for each active process 118, 120, 122 and the like. This structure will be referenced from this point on as Process Valid Address Range in the text of this document.

Process Valid Address Range List holds the address range into which the process 118, 120, 122 and the like was loaded to. Process Valid Address Range also holds all the address ranges into which diverse Dynamic Link Library (DLL) 130, 136, 142 and the like were loaded. Dynamic Link Library is a set of callable subroutines linked as a binary image that can be dynamically loaded by applications that utilize them.

Finally, SFE Server will insert API Interception Module 134, 140, 146 and the like to all active processes 118, 120, 122 and the like (step 19) as described in detail in Israel Patent Application No. XXXXXXX.

SFE Server operation of monitoring system calls in step 152

of Fig. 2 is described in detail in Israel Patent Application No. XXXXXXX.

Turning now to Fig. 4 which is a high-level flow diagram of SFE Server or the like response to an intercepted system call, referred to as step 154 of Fig. 2. SFE Server 116 determines in step 160 whether the system call detected is an illegal call or a legal call. SFE Server determines whether said system call is valid by comparing said system call originating address with range of Process Valid Address associated with said process from which said system call originated. If illegal call was detected the SFE Server 116 may terminate the illegal function (step 164). Alternatively SFE Server 116 may notify a user (typically the System Administrator) about the illegal call (step 166). Alternatively, SFE Server 116 may perform another or other series of user predetermined actions.

If the system call detected is legal (step 160) SFE Server 116 examines the said system call to determine if it is of the type of process creation (step 162). If and when it is determined that the system call of the type of process creation SFE Server 116 inserts API Interception Module 134 to the newly created process address space 118 (step 168) and updates Process Valid Address Range List (step 170) by adding

said process address list to Process Valid Address Range. If the decision in step 162 is negative SFE Server optionally performs any other user predetermined or instructed action (step 166). If and when it is determined that the system call of the type of process termination SFE Server 116 updates Process Valid Address Range List (step 171) by removing said process valid addresses range from Process Valid Address Range List.

SFE Server operation of monitoring library calls in step 156 of Fig. 2 is described in detail in US Patent Application No. XXXXXXX.

Turning now to Fig. 5 which is a high-level flow diagram of the SFE Server and the like response to an intercepted library call referred to as step 158 of Fig. 2. SFE Server 116 determines in step 172 if the library call detected is an illegal call. SFE Server determines whether said library call is valid by comparing said library call originating address with range of Process Valid Address associated with said process from which said library call originated. If an illegal call is detected SFE Server 116 optionally terminates the illegal library function (step 180). Alternatively, SFE Server 116 notifies a user (typically the System Administrator) (step 182). Alternatively, SFE

Server 116 performs any other user predetermined or instructed action (step 182).

If the library call detected is legal (step 172) SFE Server 116 determines if the said library call is of the type of DLL 130 load (step 174). If the decision in step 174 is affirmative than SFE Server 116 hooks and patches the library calls (APIs) 132 existing within said loaded DLL 130. Such hooking and patching is further described in detail in Israel Patent Application _____. After hooking and patching said API Interception Module 134 already loaded into said associated process 118 is now operative to intercept calls made to said library calls 132. When determined that the library call is of the type DLL load SFE Server 116 updates Process Valid Address Range List (step 178) by adding DLL address rage into Process Valid Address Range List. If the decision in step 172 is negative SFE Server 116 determines if the intercepted library call if of the type DLL unload (step 176) by deleting DLL address range from Process Valid Address Range List. When it is determined that the library call is of the type of DLL unload SFE Server updates the Process Valid Address Range List (step 178).

Reference in now made to Fig. 6 that is a high-level flow diagram of the operation of the Calling Address Validation Routine

module. The Calling Address Validation Routine module may operate in conjunction with API Interception Module Pre-Entry routine as further described in detail in Israel Patent Application No. XXXXXXX.

Pre-Entry routine may be activated when an API 132 or the like of Fig. 1 is intercepted. Operating under SFE System 100 Pre-Entry routine, Calling Address Validation Routine module is executing a set of instructions designed to validate the API function 132 of Fig. 1 calling address (caller Routine). Caller Routine also includes caller Application Program Interface, caller system call, caller library call and the like.

Calling Address Validation Routine module commences its operation by reading the caller Routine return address from the Procedure Activation Record (stack frame) which is on the user stack segment (step 191). The stack frame is a dynamic area of the process stack segment used as a control area for function calls. The process stack segment is a dynamic area of memory belonging to a process. In step 192 the caller Routine calling address is calculated (step 192) and with the help of the data in Process Valid Address Range List it is examined if the said caller Routine calling address is within valid address range limits (step 194). In step 196 it is determined whether the calling address valid or non-valid. To calculate if said caller Routine calling address is within said valid address range limit said caller

Routine calling address is matched with said valid address range limit. If said caller Routine calling address is within said valid address range than caller Routine calling address is valid. Next, Calling Address Validation Routine module by Pre-Entry routine or the like notifies SFE Server 116 or the like about the test result (step 198 and step 200).

It will be appreciated to by persons skilled in the art that in this illustrated embodiment of the present invention any unauthorized or illegal system call or library call originating from memory areas out of active process address space memory device 118, 120, 122 and the like of Fig. 1 will be detected and optionally their execution will be prevented by SFE System 100.

Reference is now made to Fig. 7 which is a high-level flow diagram of the Calling Address Validation Routine module relating to an another embodiment of the present invention.

In the embodiment thereof Calling Address Validation Routine module commences its operation by reading the caller return address from the Procedure Activation Record (stack frame) on the process stack segment (step 202). It will be appreciated that reading caller Routine return address is significantly faster and more accurate. In step 204 the caller Routine calling address is calculated and it is examined

with the help of system-level structures to determine whether the calling address is inside the address limits of the process stack segment (step 206). Such determination is accomplished by comparing said caller Routine calling address with address limits of said process stack segment. Next, Calling Address Validation Routine module by Pre-Entry routine or the like notifies SFE Server 116 or the like about the result of the examination (step 210 and step 212).

It will be appreciated by persons skilled in the art that in this further embodiment of the invention any unauthorized or illegal system call or library call originating from the process stack segment structure will be detected and optionally prevented by SFE System 100.

Additional advantages will readily occur to the person skilled in the art. The invention, in its broader aspects is, therefore, not limited to the specific details, representative methods, systems and examples shown and described. It will be further appreciated by persons skilled in the art that the present invention is not limited to what has been particularly shown and described hereinabove. Rather the scope of the applicant's general inventive concept and the claims which follow.

CLAIMS

1. In a computer system running an operating system platform, said operating system including a kernel space and a process space, said process space including a user application running in process space, said user application operative to intercept system calls, a method of secure function execution, said method comprising the step of:

   examine said intercepted system call validity by comparing said intercepted system call originating address with range of process valid addresses associated with said process from which said intercepted system call originated.

2. The method of claim 1, further comprising the step of:

   providing notification as to the validity of said intercepted system call.

3. The method of claim 1, further comprising the step of:

   terminating said intercepted system call.

4. The method of claim 2, further comprising the step of:

terminating said intercepted system call.

5. The method of claim 1, further comprising the steps of:

responsive to process creation inserting application program interface interception module into said created process;

responsive to process creation updating process valid addresses table.

6. The method of claim 1, further comprising the step of:

responsive to process termination updating process valid addresses table;

7. In a computer system running an operating system platform, said operating system including a kernel space and a process space, said process space including a user application running in process space, said user application operative to intercept library calls, a method of secure function execution, said method comprising the step of:

examine said intercepted library call validity by comparing said intercepted library call originating address with range of process valid addresses associated with said process from

which said intercepted library call originated.

8. The method of claim 7, further comprising the step of:

providing notification as to the validity of said intercepted library call.

9. The method of claim 7, further comprising the step of:

terminating said intercepted library call.

10. The method of claim 8, further comprising the step of:

terminating said intercepted library call.

11. The method of claim 7, further comprising the steps of:

responsive to system call loading dynamic link library hooking and patching library routines associated with said dynamic link library;

responsive to system call unloading dynamic link library updating process valid addresses table;

12. In a computer system running an operating system platform, said operating system including a kernel space and a process space,

said process space including a user application running in process space, said user application operative to system and function calls, said system or function call intercepted, a method of secure function execution, said method comprising the steps of:

receiving caller routine return address from said process memory device;

determining whether caller routine address is valid by comparing said caller address routine with process valid address table.

13. The method of claim 12, further comprising the step of:

providing notification as to the validity of said caller routine return address.

14. The method of claim 12, further comprising the step of:

performing user predetermined acts associated with said validity of caller routine address.

15. The method of claim 12, wherein the step of receiving caller routine return address from said process memory device further comprises the step of:

determining said caller routine calling address by determining the

address preceding said caller routine address.

16. In a computer system running an operating system platform, said operating system including a kernel space and a process space, said process space including a user application running in process space, said user application operative to system and function calls, said system or function call intercepted, a method of secure function execution, said method comprising the steps of:

receiving caller routine return address from said process memory device;

determining whether caller routine address is valid by comparing said caller address routine with associated process stack address area.

17. The method of claim 16, further comprising the step of:

providing notification as to the validity of said caller routine return address.

18. The method of claim 16, further comprising the step of:

performing user predetermined acts associated with said validity of caller routine address.

19. The method of claim 16, wherein the step of receiving caller routine return address from said process memory device further comprises the step of:

determining said caller routine calling address by determining the address preceding said caller routine address.

20. The method of secure function execution as substantially described hereinabove.

21. The method of secure function execution as illustrated in any of the drawings.

For the Applicant

Soroker – Agmon, Law Offices

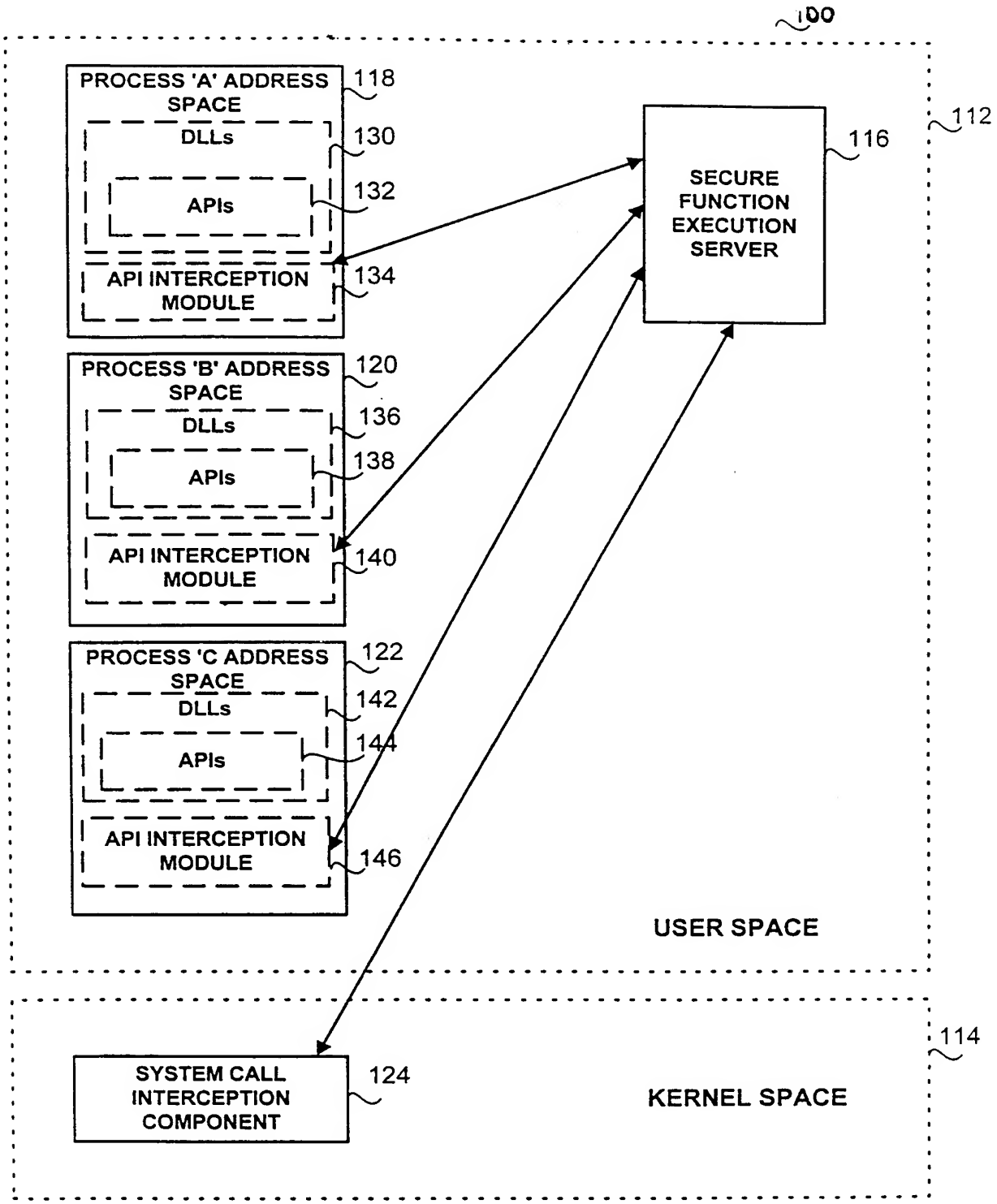

100
PROCESS 'A' ADDRESS SPACE
118
DLLs
130
APIs
132
API INTERCEPTION MODULE
134
PROCESS 'B' ADDRESS SPACE
120
DLLs
136
APIs
138
API INTERCEPTION MODULE
140
PROCESS 'C ADDRESS SPACE
122
DLLs
142
APIs
144
API INTERCEPTION MODULE
146
SECURE FUNCTION EXECUTION SERVER
116
112
USER SPACE
SYSTEM CALL INTERCEPTION COMPONENT
124
KERNEL SPACE
114

FIG. 1

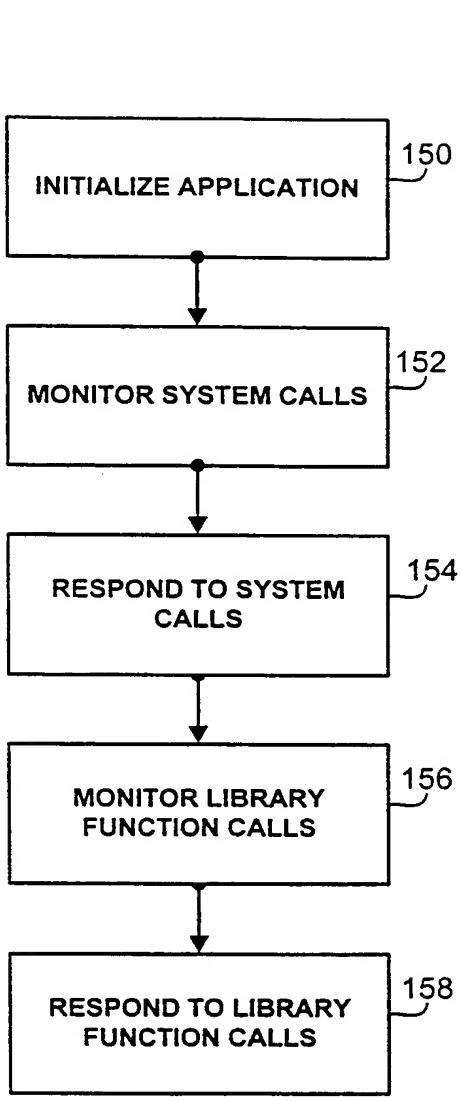

116
INITIALIZE APPLICATION
150
MONITOR SYSTEM CALLS
152
RESPOND TO SYSTEM CALLS
154
MONITOR LIBRARY FUNCTION CALLS
156
RESPOND TO LIBRARY FUNCTION CALLS
158

FIG. 2

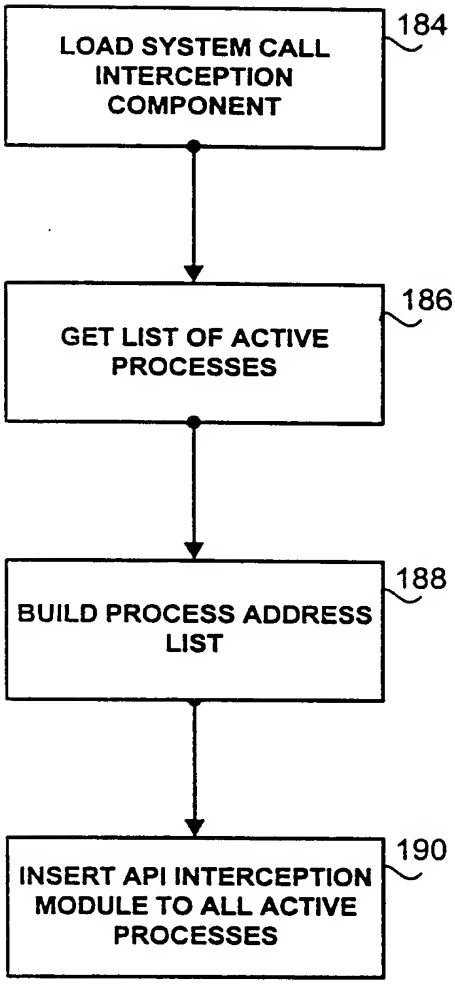

LOAD SYSTEM CALL INTERCEPTION COMPONENT
184
GET LIST OF ACTIVE PROCESSES
186
BUILD PROCESS ADDRESS LIST
188
INSERT API INTERCEPTION MODULE TO ALL ACTIVE PROCESSES
190

FIG. 3

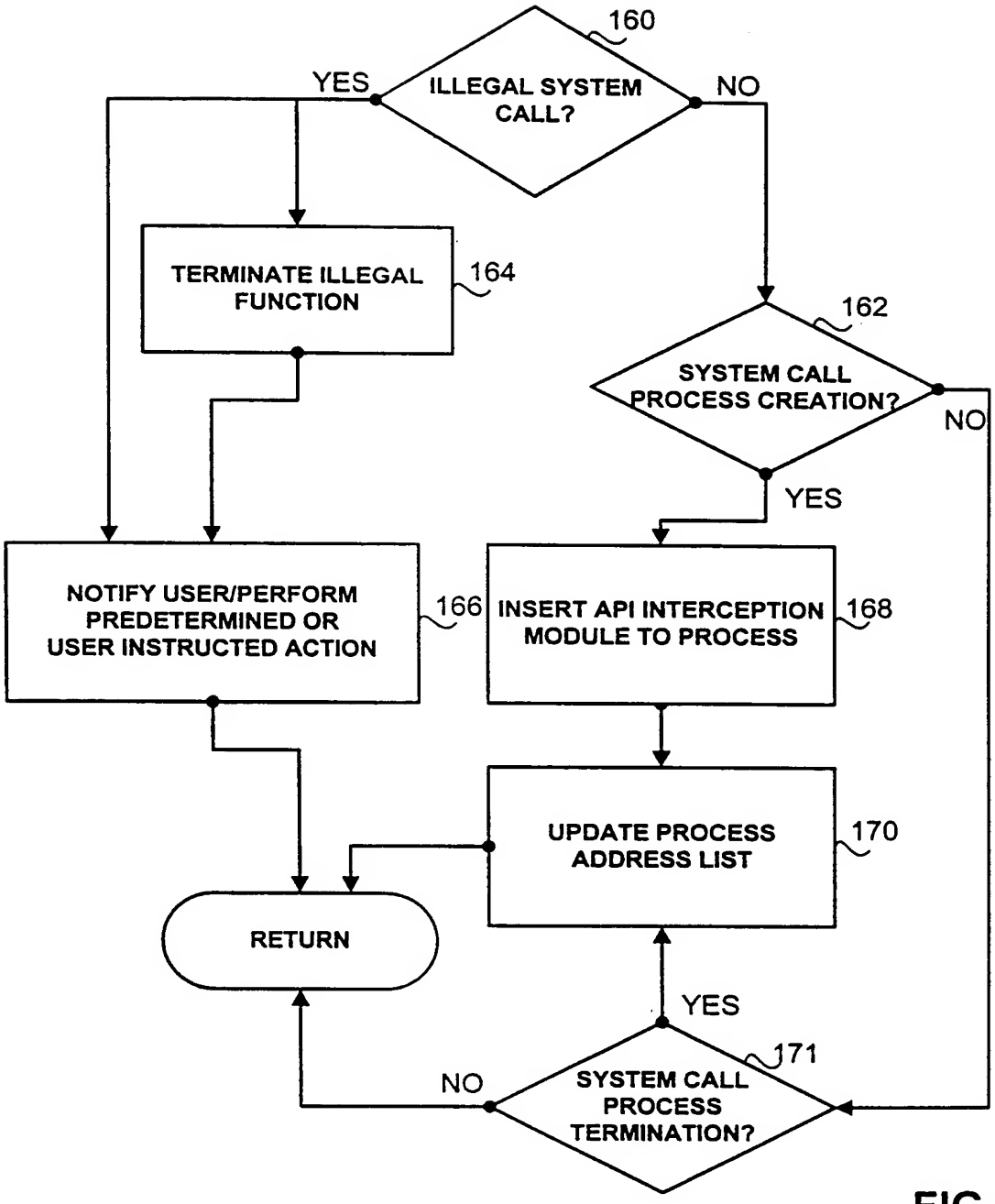

160
ILLEGAL SYSTEM CALL?
YES
NO
TERMINATE ILLEGAL FUNCTION
164
162
SYSTEM CALL PROCESS CREATION?
NO
YES
NOTIFY USER/PERFORM PREDETERMINED OR USER INSTRUCTED ACTION
166
INSERT API INTERCEPTION MODULE TO PROCESS
168
UPDATE PROCESS ADDRESS LIST
170
RETURN
YES
171
NO
SYSTEM CALL PROCESS TERMINATION?

FIG. 4

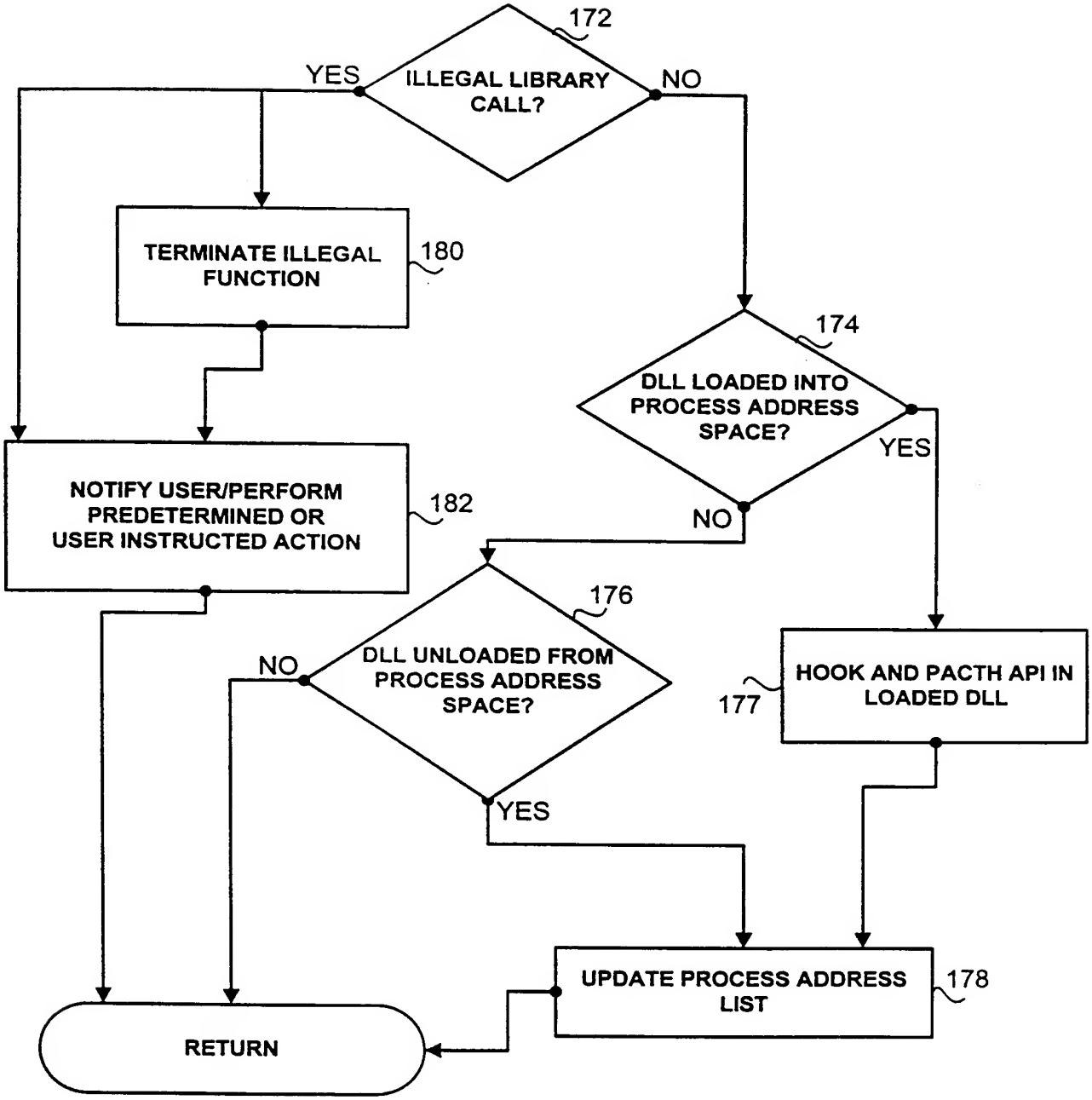

172
ILLEGAL LIBRARY CALL?
YES
NO
TERMINATE ILLEGAL FUNCTION
180
174
DLL LOADED INTO PROCESS ADDRESS SPACE?
YES
NO
NOTIFY USER/PERFORM PREDETERMINED OR USER INSTRUCTED ACTION
182
176
DLL UNLOADED FROM PROCESS ADDRESS SPACE?
NO
YES
177
HOOK AND PACTH API IN LOADED DLL
UPDATE PROCESS ADDRESS LIST
178
RETURN

FIG. 5

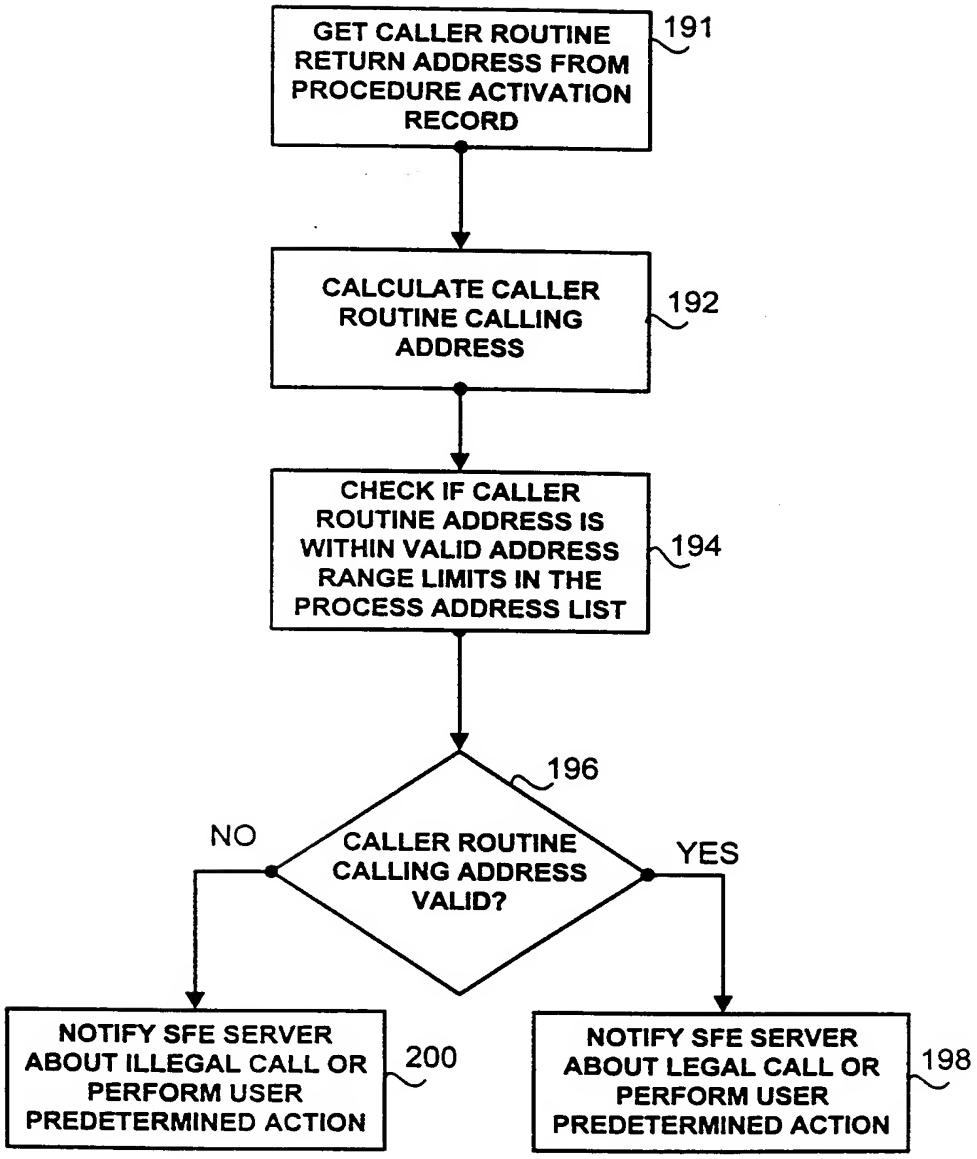

GET CALLER ROUTINE RETURN ADDRESS FROM PROCEDURE ACTIVATION RECORD
191
CALCULATE CALLER ROUTINE CALLING ADDRESS
192
CHECK IF CALLER ROUTINE ADDRESS IS WITHIN VALID ADDRESS RANGE LIMITS IN THE PROCESS ADDRESS LIST
194
196
CALLER ROUTINE CALLING ADDRESS VALID?
NO
YES
NOTIFY SFE SERVER ABOUT ILLEGAL CALL OR PERFORM USER PREDETERMINED ACTION
200
NOTIFY SFE SERVER ABOUT LEGAL CALL OR PERFORM USER PREDETERMINED ACTION
198

FIG. 6

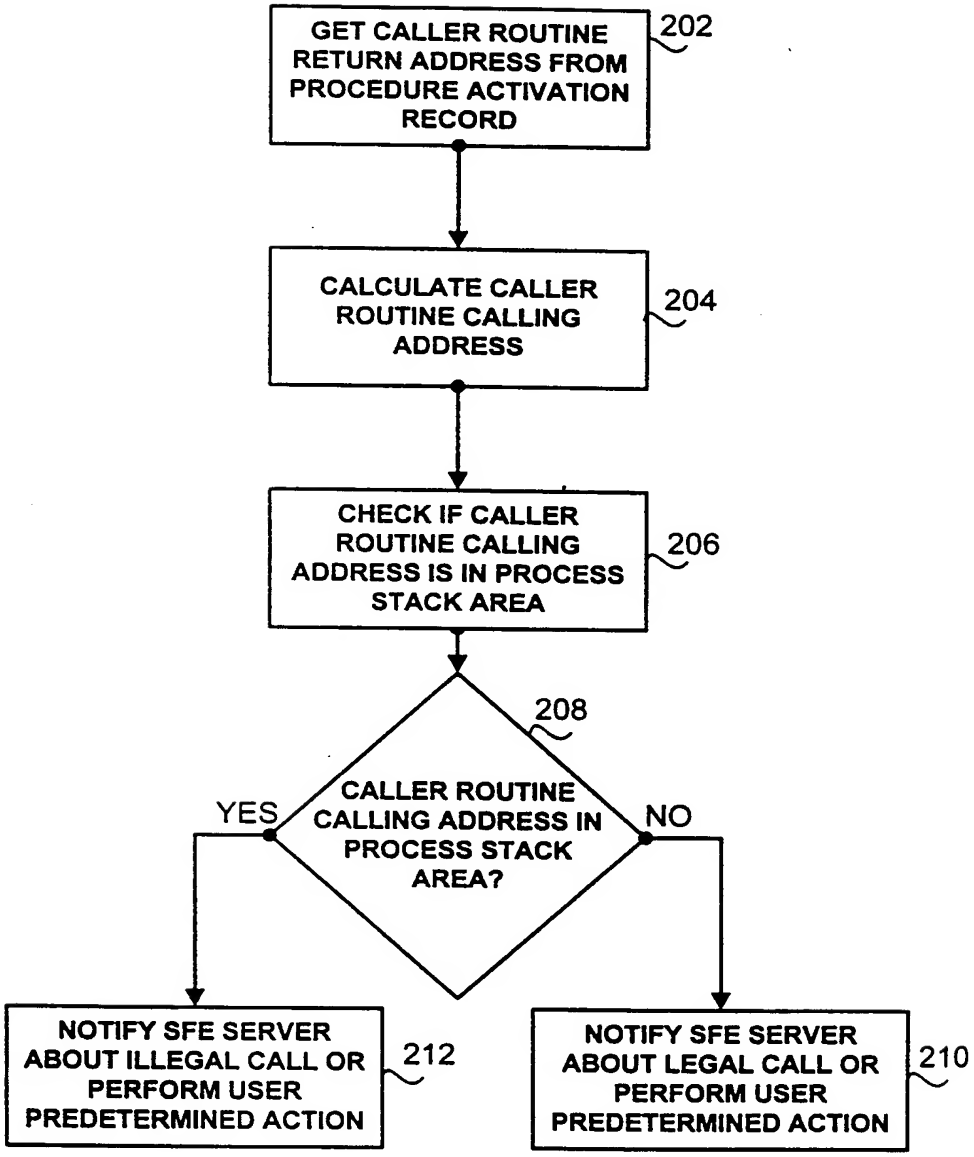

GET CALLER ROUTINE RETURN ADDRESS FROM PROCEDURE ACTIVATION RECORD
202
CALCULATE CALLER ROUTINE CALLING ADDRESS
204
CHECK IF CALLER ROUTINE CALLING ADDRESS IS IN PROCESS STACK AREA
206
208
CALLER ROUTINE CALLING ADDRESS IN PROCESS STACK AREA?
YES
NO
NOTIFY SFE SERVER ABOUT ILLEGAL CALL OR PERFORM USER PREDETERMINED ACTION
212
NOTIFY SFE SERVER ABOUT LEGAL CALL OR PERFORM USER PREDETERMINED ACTION
210

FIG. 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/31032

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06F 11/30, 12/14, 15/173, 15/177; H04L 9/00, 9/32
US CL : 713/200, 201; 709/217, 218, 219, 223, 224, 225
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 713/200, 201; 709/217, 218, 219, 223, 224, 225

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X,E	US 6,185,689 B1 (TODD, SR. ET AL) 06 FEBRUARY 2001, SEE ENTIRE DOCUMENT	1-21
X,P	US 6,088,804 A (HILL ET AL) 11 JULY 2000, SEE ENTIRE DOCUMENT	1-21
X,P	US 6,067,620 A (HOLDEN ET AL) 23 MAY 2000, SEE ENTIRE DOCUMENT	1-21
X,P	US 6,061,798 A (COLEY ET AL) 09 MAY 2000, SEE ENTIRE DOCUMENT	1-21
X	US 5,940,591 A (BOYLE ET AL) 17 AUGUST 1999, SEE ENTIRE DOCUMENT	1-21
X	US 5,832,228 A (HOLDEN ET AL) 03 NOVEMBER 1998, SEE ENTIRE DOCUMENT	1-21

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 25 FEBRUARY 2001	Date of mailing of the international search report 05 APR 2001
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer CHRISTOPHER A ~~REVAK~~ R. Matthews Telephone No. (703) 305-9618

Form PCT/ISA/210 (second sheet) (July 1998)★

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/31032

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5,826,014 A (COLEY ET AL) 20 OCTOBER 1998, SEE ENTIRE DOCUMENT	1-21
X,P	US 5,577,209 A (BOYLE ET AL) 19 NOVEMBER 1996, SEE ENTIRE DOCUMENT	1-21

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/31032

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

BRS (FILES: USPAT, DERWENT, JPO, EPO, IBM TDB'S), DIALOG (FILES: COMPSCI, ELECTRON, SOFTWARE)

search terms: buffer, overflow, attack, intrusion, hack, hacker, denial, service, flood, packet, valid, validity, validation, confirm, confirmation, confirming, intercept, interception, intercepting, intercepted, address, addresses, location, originate, originating, originated